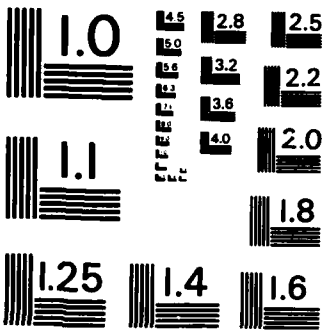
AD-A146 217 NEW LABELING PROCEDURES FOR THE BASIS GRAPH IN GENERALIZED NETWORKS(U) TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES M ENGQUIST ET AL. JUN 84 CCS-468 1/1

UNCLASSIFIED N00014-81-C-0236 F/G 12/1 NL

END

FILMED

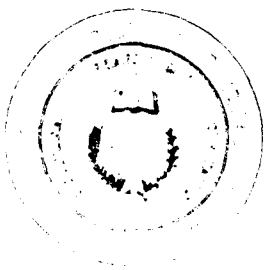
DTIC



(12)

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712



Research Report CCS 468

NEW LABELING PROCEDURES FOR THE BASIS GRAPH IN GENERALIZED NETWORKS

by

Michael Engquist

Michael D. Chang

June 1984

This research was supported in part by ONR Contract N00014-81-C-0236 and USARI Contract MDA903-83-K-0312 with the Center for Cybernetic Studies, The University of Texas at Austin. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Graduate School of Business 4.138
The University of Texas at Austin
Austin, Texas 78712
(512) 471-1821

ABSTRACT

Labeling procedures for the basis graph of a generalized network are introduced which build on procedures designed for pure networks. The various cases which arise in updating the basis graph are presented, and the efficiency of the related primal simplex implementation is discussed.

KEY WORDS

Network Flow Algorithms

Generalized Networks

Primal Simplex Method

1. INTRODUCTION

Generalized network problems form a special class of linear programming (LP) problems. A wide variety of important planning problems can be modeled as generalized networks in the areas of scheduling, cash management, production, and distribution as well as many others.

Glover et al. [8] report that their generalized network code NETG is fifty times more efficient than a general purpose LP code, and cost savings of forty to one are reported in [5] when NETG is used instead of a general purpose code. Although such special purpose generalized network optimization codes are already extremely fast, the size and importance of many generalized network problems indicates a need for further improvements in efficiency. In this paper we present a method for implementing the primal simplex algorithm for generalized networks which builds on the labeling procedures used in one of the fastest pure network codes [2]. Our approach has the advantage of being quite easy to understand, and it should give rise to computer codes which are highly efficient and easily maintained.

Any LP problem whose coefficient matrix, not including simple bounding constraints, contains at most two nonzero entries per column is a generalized network problem. Lower bounds on variables are translated out and the resulting upper bounds are known as capacities. By scaling or by complementing a variable relative to its capacity, these problems can be transformed so that each column has at least one entry which is +1. A graphical representation of these problems as directed networks is obtained as follows. Each row of the transformed matrix is associated with a node. Each column of this matrix is associated with an arc directed away from the node corresponding to the +1 column entry (arcs corresponding to columns containing two +1 entries can be directed either way). Columns with a single nonzero entry are

associated with arcs incident on only one node, and such arcs are called self-loops. Thus, a column with nonzero components of +1 and -m in rows i and j, respectively, corresponds to the generalized arc (i,j) of Figure 1. The value m, which is shown in the triangle, is called the multiplier of the arc.

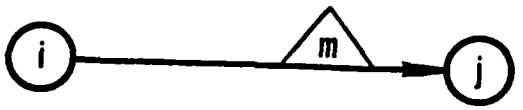


Figure 1. A generalized arc.

The variable x associated with the arc of Figure 1 can be interpreted as an amount of flow leaving node i. These x units of flow are transformed along the arc to mx units of flow entering node j.

An efficient implementation of the primal simplex algorithm for generalized network problems was developed by Elam, Glover, Hultz, Klingman, and Stutz [6], [8], [9], [10]. Their method for maintaining the basis graph is known as the Extended Augmented Predecessor Index method, and it is based on the triple label representation for trees introduced by Johnson [12]. Glover, Klingman, and Stutz [11] utilized a node function called the thread in their Augmented Threaded Index method for pure networks. Bradley, Brown, and Graves [3] also make use of the thread in the solution of pure network problems. Recently, the thread was used in primal simplex implementations for generalized networks by Adolphson [1] and by Brown and McBride [4]. Our approach also uses the thread but our methods differ from previous work in that we do

not make use of the "rooted loop orientation" [6], [10] in our quasi-tree representation.

Our method of representing the basis should prove useful when applied to processing networks [7], [13] in which generalized arcs are present. It is also possible that our approach will lead to an extension to generalized networks of some work reported in [14]. There, a way of implementing tree operations in pure networks is given for which a time bound of O(log n) per tree operation results, where n is the number of nodes.

2. BASIS STRUCTURE

It is well known that a basis for a generalized network problem can be represented graphically as a collection of quasi-trees, where a quasi-tree is a tree to which a single arc has been adjoined. Thus, a quasi-tree contains a single closed path, and this closed path is called a loop. It is possible for this loop to be a self-loop.

Specialized implementations of the primal simplex algorithm for generalized network problems rely on functions defined on the nodes of the basis graph. The node functions we use are those of Barr, Glover, and Klingman [2], and we adopt their notation whenever possible. However, the node functions of [2] apply to rooted trees rather than quasi-trees so we must set the stage properly. Suppose that a quasi-tree Q is given. We select one of the arcs of the loop of Q and designate it as the special arc of Q. When the special arc is removed from Q, a tree T results. One of the end nodes of the special arc is designated as the root of T. The root node is regarded pictorially as the highest node of T with all other nodes hanging below it. If i and j are end nodes of an arc of T such that i is closer to the root than j, then i is called the predecessor of j and node j is called an

immediate successor of node i. The subtree of T which contains a node x and all of its successors will be denoted as T(x). The node functions we need are defined as follows.

For nodes x other than the root node, p(x) denotes the predecessor of x. If x is the root node, then p(x) is the node at the opposite end of the special arc.

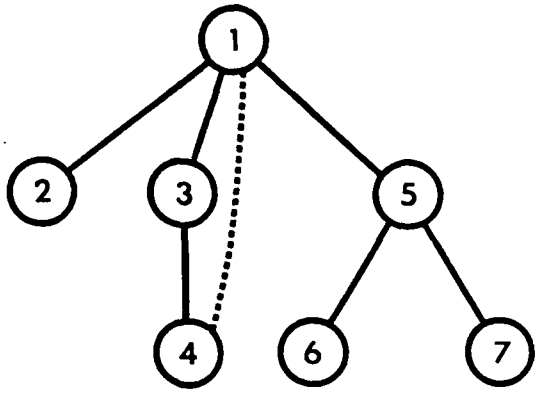
The preorder thread index (thread, for short) of node x is denoted as s(x). A preorder of T is an order on its nodes in which node y precedes its successors and these successors precede any other node which follows node y. Once a preorder is given, the thread is determined as the function which traces the tree nodes in preorder. The value of the thread for the last node in the preorder is defined to be the root.

The function whose value at node x is the number of nodes in T(x) is denoted by t(x).

The last node of T(x) in the preorder defining s is denoted by f(x).

The backpath of a node x in T consists of node x and all nodes obtained from x by successively applying the predecessor until the root is reached. Let z denote the end node of the special arc which is opposite to the root. In order to carry out the primal simplex method, we require that the predecessor values be flagged negative on the backpath of z.

An example of a quasi-tree with the structures we have defined is shown in Figure 2. In this figure, we have used undirected arcs (links).



	Functions			
Node	p	s	f	t
1	-4	2	7	7
2	1	3	2	1
3	-1	4	4	2
4	-3	5	4	1
5	1	6	7	3
6	5	7	6	1
7	5	1	7	1

Figure 2. Example quasi-tree (special arc dotted).

3. PRIMAL SIMPLEX SUMMARY

In this section we briefly outline the steps of the simplex method specialized to minimum cost generalized network problems. We limit the details of actual calculations since they are available elsewhere, e.g. [4], [6], and we wish to highlight the use of the node functions introduced in Section 2.

The node potential (dual variable) at node k is denoted as π_k. The node potential of the root node is set first. For this purpose, we take successive predecessors until a complete trace of the loop nodes is made. Once the node potential of the root is obtained, a trace of the entire tree is made by following the thread in order to set the remaining π_k values. If $j = p(k)$, then π_k is determined from π_j by means of a standard calculation.

The incoming arc is chosen either as a nonbasic arc with zero flow and negative reduced cost or as a nonbasic arc with flow at capacity and positive reduced cost. If an incoming arc cannot be found, the problem is solved.

The outgoing arc is determined by tracing predecessors back from the end nodes u and v of the incoming arc until a node k with $p(k) < 0$ is reached or until a point of intersection of these predecessor paths is reached. The latter situation is detected by moving back along the predecessor path from u to x_u and along the predecessor path from v to x_v until the condition $t(x_u)=t(x_v)$ holds. At this point either $x_u=x_v$ or x_u and x_v are replaced by taking further predecessors. If a point of intersection (called the join) is reached, the predecessor paths coincide past this point. Ultimately, one or two nodes k with $p(k) < 0$ are reached. Such nodes are on one or possibly two quasi-tree loops. Next, the loop or loops must be traced by again following the predecessor. As the predecessor paths of u and v are traced, the basis representation of the entering arc is computed and the ratio test is carried out to determine the outgoing arc (p,q). The predecessor paths of u and v must be retraced to accomplish the flow change.

In the basis exchange step the incoming arc (u,v) becomes basic, the outgoing arc (p,q) becomes nonbasic, and the node functions and other structures defined in Section 2 are updated. To facilitate the discussion in Section 4, we shall replace the directed arc (u,v) by the undirected arc or link [u,v]. Likewise, (p,q) is replaced by [p,q].

4. BASIS EXCHANGE

For a quasi-tree Q, the link corresponding to the special arc is denoted as [x,y]. As usual, T will denote the tree obtained from Q when [x,y] is removed. The tree T is assumed to be rooted at x and to be equipped with the node functions p, s, t, and f. For the outgoing link [p,q], it will be assumed that node p is the predecessor of node q. The updating procedures for the node functions will be given

as five mutually exclusive cases based on conditions involving [u,v] and [p,q]. These conditions can be checked and the appropriate case determined during the ratio test.

Before proceeding with the cases, we introduce some operations.

setsp([x,y]): the special arc of Q is set by letting p(x) equal y.

remsp([x,y]): the special arc is removed from Q by setting p(x)=0. This predecessor setting for the root node conforms to the definition given in [2].

negpath([x,y]): the predecessor values of nodes on the backpath of y in T are negated.

split([p,q]): the link [p,q] is removed from T leaving two trees, T(q) rooted at q and T-T(q) with the original root. The node functions are updated. The details of this update are found in Section 3.2, Step I of [2].

In the next operation, it is assumed that S is some rooted tree which contains node z.

reroot(S,z): the predecessor orientation on the backpath of z in S is reversed making z the root and the other node functions are updated appropriately. The details of this updating operation are found in Section 3.2, Step III of [2].

For the last operation, it is assumed that S and W are two rooted trees

and that a or b is some node of S while the remaining node is the root of W.

attach([a,b]): a new rooted tree is created by connecting W to S via [a,b]. The root of S serves as the root of the new tree, and the node functions are updated. The details for the update are found in Section 3.2, Step IV of [2].

Next, we consider the cases which come up in updating the basis graph. In the first three cases, we assume that both [u,v] and [p,q] have their end nodes in the same quasi-tree Q. To avoid trivial cases, we assume that [u,v] $\neq$ [p,q].

Case 1: Link [p,q] is on the original loop of Q.
Apply remsp([x,y]) and negpath([x,y]). If [p,q] =[x,y], perform reroot(T,u), setsp([u,v]), negpath([u,v]), and stop. Otherwise, apply split([p,q]). Let T_1 contain u where T_1 is either T(q) or T-T(q), and define T_2 as the subtree remaining. Let w be the end node of [x,y] in T_2. Next, reroot(T_1,u), reroot(T_2,w) and attach([x,y]). Finally, apply setsp([u,v]), negpath([u,v]) and stop.

Case 2: Link [p,q] is on the loop determined by [u,v] but not on the original loop of Q.
First, apply split([p,q]). Let w be the end node of [u,v] in T(q). Then apply reroot(T(q),w), attach([u,v]) and stop.

Case 3: Link [p,q] is neither on the loop determined by [u,v] nor on the original loop of Q.

Apply split([p,q]). Then perform reroot(T(q),u), setsp([u,v]), negpath([u,v]), and stop.

In the next two cases, we assume that [u,v] connects quasi-trees Q_L and Q_R with u in Q_L and v in Q_R. Also it is assumed that p and q are in Q_L.

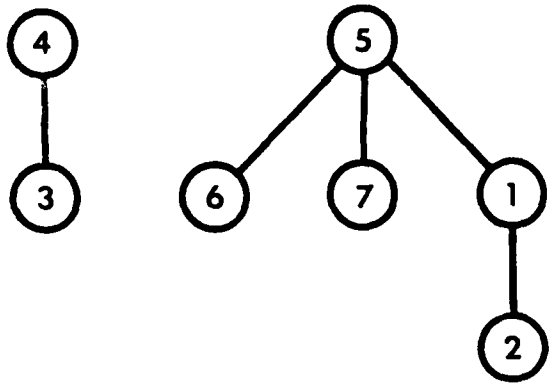
Case 4: Link [p,q] is not on the loop of Q_L.
Apply split([p,q]). Then reroot(T_L(q),u), attach([u,v]), and stop.

Case 5: Link [p,q] is on the loop of Q_L.
Apply remsp([x_L,y_L]) and negpath([x_L,y_L]). If [p,q]=[x_L,y_L], perform reroot(T_L,u), attach([u,v]), and stop. Otherwise, split([p,q]). Let T_1 contain u, where T_1 is either T_L(q) or T_L-T_L(q), and define T_2 as the subtree remaining. Apply reroot(T_1,u) and attach([u,v]). Let w be the end node of [x_L,y_L] in T_2. Apply reroot(T_2,w), attach([x_L,y_L], and stop.

As an example we let u=5, v=2, p=1, and q=3 in Figure 2, so that Case 1 applies. Node w is 4 and Figure 3 shows the trees which result when all steps of Case 1 up to and including reroot(T_1,5) and reroot(T_2,4) are applied. Finally, the quasi-tree obtained when the remaining steps of Case 1 are applied is shown in Figure 4.

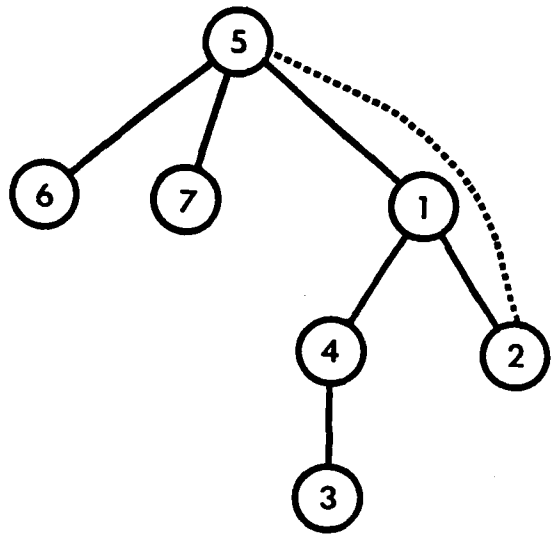
5. COMPUTATIONAL CONSIDERATIONS

In the case of pure networks, node potentials can be updated on either of the two subtrees produced when the leaving arc is removed from the basis tree. In [2], the subtree with the smaller number of nodes is identified using t(x) values, and the node potentials are updated on this subtree. For generalized networks, however, the node



	Functions			
Node	p	s	f	t
1	5	2	2	2
2	1	5	2	1
3	4	4	3	1
4	0	3	3	2
5	0	6	2	5
6	5	7	6	1
7	5	1	7	1

Figure 3. Trees after rerooting.



	Functions			
Node	p	s	f	t
1	-5	4	2	4
2	-1	5	2	1
3	4	2	3	1
4	1	3	3	2
5	-2	6	2	7
6	5	7	6	1
7	5	1	7	1

Figure 4. Quasi-tree after updating (special arc dotted).

potentials are uniquely determined and consequently their update does not play a prominent role in the organization of the updating procedures. Nevertheless, whenever a loop involved in the basis exchange remains intact, as in Cases 2, 4 and 5 of Section 4, node potentials are updated only on the subtrees which are attached to the quasi-tree containing the loop. This can be carried out by tracing these subtrees using the thread.

Careful attention to the lengths of backpaths involved in rerooting subtrees should pay off in reduced updating times in Cases 1 and 3.

In Case 1, when u and v are contained in different subtrees after [p,q] is removed, we may assume, without loss of generality, that u is in T-T(q). When this situation applies, and the length of the backpath of v in T(q) is less than the combined backpath lengths of y in T(q) and u in T-T(q), then the alternative reroot(T(q),v) should replace the rerooting operations previously given with v becoming the root of T.

Suppose now that u and v are in a single subtree after [p,q] is removed in Case 1 or in Case 3 and let this subtree be S. If the backpath of v in S is shorter than the backpath of u in S, then reroot(S,v) should replace reroot(S,u).

Another approach which may prove to be effective in streamlining Cases 1 and 3 involves the introduction of a new node function b. When b is used, the special arc no longer needs to have one of its end nodes as the root. For nodes y on the path from a selected end node of the special arc to the root, b(y) is set to x in case y=p(x). On the remainder of the loop, b is set to a large integer value M, and it is set to zero elsewhere. In determining the leaving arc, one traces a path using predecessors from an end node of the incoming arc until a node x satisfying $b(x) \neq 0$ is reached. Then b is used to trace one side of the loop and p is used to trace the other side. The use of b allows a rerooting operation in Cases 1 and 3 to be eliminated; however, in

Case 5, when the root of T_L is in T_2, one of the rerooting operations will involve more work when b is used. In general, the values of b must be updated, and tracing the loop is somewhat more difficult when b is used. An experimental FORTRAN code for solving generalized networks has been written which incorporates the b function, but the limited testing to date has not allowed any conclusions to be drawn concerning the efficacy of this function.

Introduction of the reverse thread function, which is the inverse of the thread function s, allows a simplified basis update. However, in the testing done in [2], inclusion of the reverse thread resulted in only a 5 percent reduction in solution times. It seems likely that this conclusion will carry over to generalized networks as well.

Much computational testing remains to be done in order to verify the efficiency of our approach. Such work is complicated by the considerable effects that starting procedures and pricing strategies have on total solution times. It is clear, however, that careful integration of node function updates and primal simplex steps is required to produce an efficient computer code using the methods which we have outlined.

REFERENCES

1. Adolphson, D., "Design of Primal Simplex Generalized Network Codes Using a Preorder Thread Index," Working Paper, School of Management, Brigham Young University, Provo, 1982.

2. Barr, R., Glover, F., and Klingman, D., "Enhancements of Spanning Tree Labeling Procedures for Network Optimization," INFOR, Vol. 17, 1979, pp. 16-34.

3. Bradley, G., Brown, G., and Graves, G., "Design and Implementation of Large Scale Primal Transshipment Algorithms," Management Science, Vol. 24, 1977, pp. 1-34.

4. Brown, G. and McBride, R., "Solving Generalized Networks," to appear in Management Science.

5. Brooks, R., "Using Generalized Networks to Forecast Natural Gas Distribution and Allocation During Periods of Shortage," Mathematical Programming Study, Vol. 15, 1981, pp. 23-42.

6. Elam, J., Glover, F., and Klingman, D., "A Strongly Convergent Primal Simplex Algorithm for Generalized Networks," Mathematics of Operations Research, Vol. 4, 1979, pp. 39-59.

7. Engquist, M., and Chen, C., "Computational Comparison of Two Solution Procedures for Allocation/Processing Networks", to appear in a forthcoming Mathematical Programming Study.

8. Glover, F., Hultz, J., Klingman, D., and Stutz, J., "Generalized Networks: A Fundamental Computer Based Planning Tool," Management Science Vol. 24, 1978, pp. 1209-1220.

9. Glover, F., and Klingman, D., "A Note on Computational Simplifications in Solving Generalized Transportation Problems," Transportation Science, Vol. 7, 1973, pp. 351-361.

10. Glover, F., Klingman, D., and Stutz, J., "Extensions of the Augmented Predecessor Index Method to Generalized Network Problems," Transportation Science, Vol. 7, 1973, pp. 377-384.

11. Glover, F., Klingman, D., and Stutz, J., "Augmented Threaded Index Method for Network Optimization," INFOR, Vol. 12, 1974, pp. 293-298.

12. Johnson, E., "Networks and Basic Solutions," Operations Research, Vol. 14, 1966, pp. 619-623.

13. Koene, J., "Minimal Cost Flow in Processing Networks, a Primal Approach," Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, 1962.

14. Sleator, D. and Tarjan, R., "A Data Structure for Dynamic Trees," Journal of Computer and System Science, Vol. 26, 1983, pp. 362-391.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CCS 468	2. GOVT ACCESSION NO. AD A146 217	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) New Labeling Procedures for the Basis Graph in Generalized Networks		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) M. Engquist, M.D. Chang		8. CONTRACT OR GRANT NUMBER(s) N00014-81-0236 C
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Cybernetic Studies The University of Texas at Austin Austin, Texas 78712		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research (Code 434) Washington, D.C.		12. REPORT DATE June 1984 13. NUMBER OF PAGES 16
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Network Flow Algorithms, Generalized Networks, Primal Simplex Method

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Labeling procedures for the basis graph of a generalized network are introduced which build on procedures designed for pure networks. The various cases which arise in updating the basis graph are presented, and the efficiency of the related primal simplex implementation is discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)